MSDS600 Week 5 Assignment - Nathan Worsham

I worked through the XOR example given in the "XOR Classification Problem.docx", but by the time I finished I really did not understand what was being accomplished (XOR script submitted along with this document). I was confused by what exactly the XOR model was predicting because if you give it values to test with—which can only be a combination of 0 and 1, it is really just "looking up" the answer because all answers are already in the model.

```
> predict(model,x)
1 2 3 4
0 1 1 0
Levels: 0 1
```

Looking in the help file for the svm, there is an iris example that I found much more clearer in my mind than the XOR example. However both examples had one similar problem that I could not resolve: there was no example of using them to predict with values outside of the model, both used the model as the test. Once I learned if I send the same data type against it—a vector for the XOR problem or data frame for the iris data–with values that are outside of the model (not applicable to the XOR), I could actually see it predicting something:

```
slength = c(5.2)
swidth = c(3.3)
plength = c(1.35)
pwidth = c(0.3)
test.df <- data.frame(slength,swidth,plength,pwidth)
pred <- predict(model, test.df)
pred
     1
setosa
Levels: setosa versicolor virginica
```

This type of machine learning that the SVM uses is "Supervised Learning" because it "uses a known dataset (called the training dataset)" (mathworks.com, n.d.). Although in supervised learning you should provide test data in which the answers are already known, so that you can test the model to see how accurate it is. This reminded me of a video I had seen earlier: How to Build a Document Classifier in R in under 25 minutes by Tim D'Auria (n.d.). Using the R coding concepts from the video, I was able to pull out a subset–70%–of the iris data as my training data, and hold back 30% as my testing data:

Below, is the original iris data example, with my comments and changes:

Later in my testing I learned that these 2 commands where causing me confusion specifically when y was assigned to "Species", so I did not use this method

```
library(e1071)
#data(iris)
#attach(iris)
```

I find the alternative much easier to understand, x will be the data.frame iris EXCEPT for the column "Species" and y will be "Species"

```
#alternatively the traditional interface:
#x <- subset(iris, select = -Species)
#y <- Species
#model <- svm(x, y)
```

I believe this is where is where the example falls apart, testing with training data feels like no test at all.

```
#test with train data
#pred <- predict(model, x)
```

Instead I chose to take a sample of the iris data.frame (70%) to use as the training data, and keep the the remaining 30% to see how the model did. Using the sample function, random row numbers are selected to use as a subset

```
train.sample.iris <- sample(nrow(iris), ceiling(nrow(iris) * 0.7))
test.sample.iris <- (1:nrow(iris)) [- train.sample.iris]
```

Now using that sample selection to subset the iris data, model and answers, I am able to create my new model.

```
x <- iris[train.sample.iris, !colnames(iris) %in% "Species"]
y <- iris[train.sample.iris, colnames(iris) %in% "Species"]
model <- svm(x, y)
```

Taking the remaining 30% and pulling out the Species column from the test.sample.iris subset to test against and another to check the answers.

```
test.iris <- iris[test.sample.iris, !colnames(iris) %in% "Species"]
test.iris.Species <- iris[test.sample.iris, colnames(iris) %in% "Species"]
```

Now running the prediction.

```
pred <- predict(model, test.iris)
```

Finally the the accuracy of the model can be checked using a confusion matrix by adding the sum of the diagonal values divided by the total number of tests. I had never heard of a confusion matrix and it is a bit difficult to read at first, but a confusion matrix—also called a contingency table—is a table that shows how the predicted values faired against the actual values (wikipedia.com, n.d.).

```
conf.mat <- table("Predictions" = pred, Actual = test.iris.Species)
conf.mat
(accuracy <- sum(diag(conf.mat)) / length(test.sample.iris) * 100)
```

Now that I had a better understanding how to use the SVM, I decided to try the same concepts against data from schooldigger.com from Colorado schools. This site provides school rankings based on test scores along with other data, such as student/teacher ratio and ethnicity makeup of the schools. Since the SVM example we had learned up to now really is just about classification, I wanted to annonomize the data and see if it could predict the city or the district. I learned that this only worked if the data is numerical and that I would have to remove any n/a values to get it to work. There are three columns with Yes or No values. I used a spreadsheet equation `=IF(B2="No",0,IF(B2="Yes",1))` to change

these values to binary then removed the original columns. As I mentioned before I removed any identifying data columns including Rank and Zip Code, though in one trial I kept City and in another I kept District as the prediction column. I ended up creating 2 functions:

The intent of the first function is essentially the same outcome I previously did with the iris data which is to provide the function a csv input file and a prediction column. The function then just like the iris example takes a 70% sample and tests against the remaining 30% and provides the accuracy that the model was able to predict:

```
week5sampleFromSelf <- function(csv.file, testVariable){
  setwd("/Users/worshamn/Dropbox/Documents/Regis/MSDS600/week5/")
  DF <- read.csv (csv.file, header = TRUE)
  library(e1071)
  train.sample.DF <- sample(nrow(DF), ceiling(nrow(DF) * 0.7))
  test.sample.DF <- (1:nrow(DF)) [- train.sample.DF]
  x <- DF[train.sample.DF, !colnames(DF) %in% testVariable]
  y <- DF[train.sample.DF, colnames(DF) %in% testVariable]
  model <- svm(x, y)
  test.DF <- DF[test.sample.DF, !colnames(DF) %in% testVariable]
  test.DF.testVariable <- DF[test.sample.DF, colnames(DF) %in% testVariable]
  pred <- predict(model, test.DF)
  conf.mat <- table("Predictions" = pred, Actual = test.DF.testVariable)
  print(conf.mat)
  (accuracy <- sum(diag(conf.mat)) / length(test.sample.DF) * 100)
}
```

The second fuction takes two csv inputs and a prediction column. The first csv is the model. The second csv is the test set in which the function removes the prediction column for the test and then uses that data to judge the accuracy:

```
week5sampleFromModel <- function(model.csv.file, test.csv.file, testVariable){
  setwd("/Users/worshamn/Dropbox/Documents/Regis/MSDS600/week5/")
  model.DF <- read.csv (model.csv.file, header = TRUE)
  test.DF <- read.csv (test.csv.file, header = TRUE)
  library(e1071)
  #train.sample.DF <- sample(nrow(DF), ceiling(nrow(DF) * 0.7))
  #test.sample.DF <- (1:nrow(DF)) [- train.sample.DF]
  x <- model.DF[, !colnames(model.DF) %in% testVariable]
  y <- model.DF[, colnames(model.DF) %in% testVariable]
  model <- svm(x, y)
  test.DF.minusVariable <- test.DF[, !colnames(test.DF) %in% testVariable]
  test.DF.testVariable <- test.DF[, colnames(test.DF) %in% testVariable]
  pred <- predict(model, test.DF.minusVariable)
  conf.mat <- table("Predictions" = pred, Actual = test.DF.testVariable)
  print(conf.mat)
  (accuracy <- sum(diag(conf.mat)) / length(test.DF.testVariable) * 100)
}
```

Turns out trying to predict the city does not work well. Against itself (the first function) around 25% accuracy and even less accuracy when trying to predict the outcomes of a different csv: example using the elementary data to try to predict the city data of middle schools

was around 15% accurate. I had similar results with district but slightly better accuracy around 30%. Seeing that this wasn't working I decided to try a different variable. I decided to see if I could predict the "Is Title I" column. A Title I school is a lower income school that receives extra federal funding. I expected that the column "Percent Free/Discount Lunch" may very well be connected. Sure enough I received much better results (also of note is I had to change the "Is Title I" column back into Yes/No data for it to work):

First function, testing the model against 30% held back data:

Elementary:

```
           Actual
Predictions  No Yes
        No  140  25
        Yes  15 106
[1] 86.01399
```

Middle:

```
           Actual
Predictions  No Yes
        No  100  12
        Yes   4  24
[1] 88.57143
```

High:

```
           Actual
Predictions No Yes
        No  90   9
        Yes  1   7
[1] 90.65421
```

But less consistency when trying to predict other models:

Elementary Predict Middle:

```
           Actual
Predictions  No Yes
        No  227   3
        Yes 124 115
[1] 72.92111
```

Elementary Predict High:

```
           Actual
Predictions  No Yes
        No  167   5
        Yes 140  46
[1] 59.49721
```

Middle Predict Elementary:

```
           Actual
Predictions  No Yes
        No  497 248
        Yes  14 197
[1] 72.59414
```

Middle Predict High:

```
           Actual
Predictions  No Yes
        No  290  16
        Yes  17  35
[1] 90.78212
```

High Predict Elementary:

```
           Actual
Predictions  No Yes
        No  506 314
        Yes   5 131
[1] 66.6318
```

High Predict Middle:

```
           Actual
Predictions  No Yes
        No  345  62
        Yes   6  56
[1] 85.50107
```

So in my example, the best outcome I could get out of one model predicting another was Middle School data predicting the High School Title I status. Though the percentage is higher, this still is not good enough to be relied upon as you would want the model to predict as close to 100% as possible.

Reference

Mathworks.com, n.d. Retrieved from http://www.mathworks.com/discovery/supervised-learning.html
Tim D'Auria, n.d. How to Build a Document Classifier in R in under 25 minutes. Retrieved from https://www.youtube.com/watch?v=j1V2McKbkLo
Wikipedia.com, n.d. Retrieved from https://en.wikipedia.org/wiki/Confusion_matrix